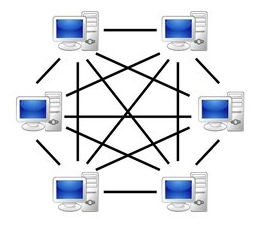
**Caso 1**

Para el caso 1 se propone utilizar una arquitectura **Peer-to-Peer** (P2P) ya que se pide que todos los nodos que integran la red puedan actuar como servidor o cliente y que estén todos sincronizados entre sí.

Además, esta arquitectura es descentralizada ya que todos los nodos son iguales.

Para conectar todas las PCs entre sí, proponemos utilizar el estilo arquitectónico **orientado a eventos** de manera que ante una transacción de un nodo, este dispare un evento y se los comunique a todos los otros nodos que intervienen. De esta manera se mantienen todos los nodos sincronizados (utilizando mensajes asincrónicos).

Con esta arquitectura, dicho pedido de cumple, el diagrama sería el siguiente:

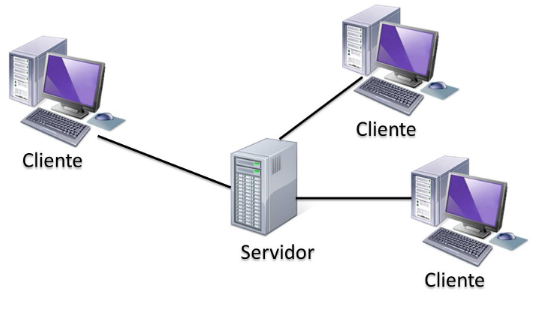


**Caso 2**

Para el caso 2 se propone utilizar una arquitectura **Cliente-Servidor** ya que al ser un servicio en donde los equipos terminales ubicados en los distintos clientes tienen la necesidad de conectarse a un único servidor central.

Por otro lado, se pide que en ningún momento la terminal quede bloqueada a la espera de una confirmación de recepción por parte del servidor de modo que, además, tolere alta frecuencia de operaciones. Esto significa que los mensajes que envían las terminales tienen que ser asincrónicos, para no quedarse esperando respuesta del servidor y poder seguir ejecutando con normalidad, y que se soporte envío de mensajes con alta frecuencia. También debe ofrecer la suficiente flexibilidad para integrar diferentes fabricantes de terminales y soportar múltiples formatos de representación de los datos. Todas estas condiciones las cumple el patrón “**bus de mensajes**”: Es un sistema de mensajería asincrónica brinda un mecanismo de transporte desacoplado, confiable y eficiente. Con soporte de envío de mensajes con alta frecuencia, transferencia inmediata y con la posibilidad de utilizar diferentes formatos de mensajes.





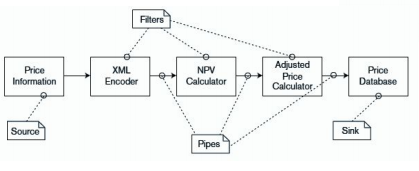
**Caso 3**

Para el caso 3 se propone utilizar por un lado una arquitectura de **Flujo de Datos** ya que se pide que cada nodo sea capaz de administrar y redireccionar datos a otros nodos de acuerdo a reglas de negocio. Además plantea la posibilidad de en algunos casos, realizar una transformación de datos para comunicárselo al siguiente nodo.

Por otro lado, se propone utilizar también el patrón **SOA**, ya que son servicios independientes que cooperan entre si con otros componentes y todos cuentan con conexión a HTTP.

Se podría usar microservicios pero se habla de sistemas y no de servicios pequeños.

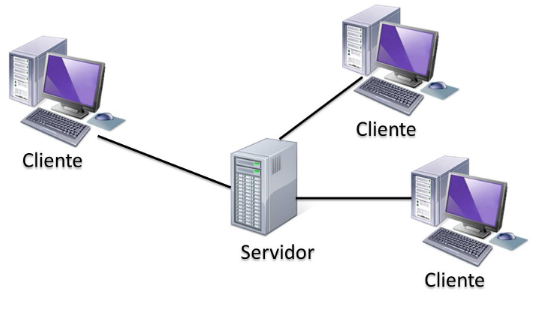
Como conectores de integración se propone utilizar call & return con protocolos SOAP o REST, dependiendo de la tecnología que manejan los servicios, ambos encuadrarían bien con lo pedido ya que se comunican mediante HTTP.



**Caso 4**

En este caso utilizaríamos el patrón **cliente-servidor**, ya que la consola sería el cliente y se indica la existencia de un servidor para recuperar los datos.

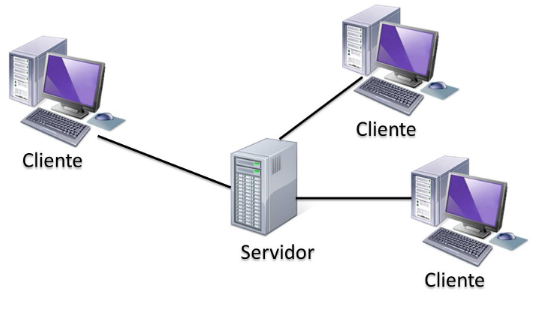
Dado que se toleran bloqueos en la consola al sincronizar la información con el servidor, podemos utilizar el patrón **call & return** para hacer la petición de los datos y que la consola se quede esperando la respuesta. La representación de los datos puede ser cualquiera elegida por el fabricante.



**Caso 5**

En este caso creemos que sería conveniente implementar una arquitectura **Cliente-Servidor** donde cada uno de los buques (clientes) tienen la necesidad de conectarse a un único servidor central (base central de la Marina).

Luego, para notificar las condiciones que atraviesan y recibir reportes, pide que se garantice la integridad de los mensajes, por eso utilizaríamos el estilo **orientado a eventos** para enviar las notificaciones con **excepciones** de forma que ante algún error, vuelva a enviar el mensaje al destino.

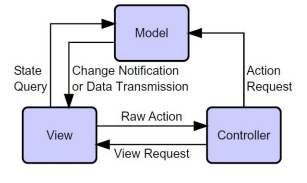


**Caso 6**

Para este último caso, ante la necesidad de contar con servicios independientes entre sí (sean de un tercero o propios), contando con una gran flexibilidad para poder adaptarlo según lo desee cada cliente, creemos beneficioso implementar una **Arquitectura MVC** donde la **capa** de modelo se diseñe **Orientada a Servicios** con conectores Call&Return **REST**.

De esta manera contamos con un protocolo estándar de los más habituales del mercado para permitir una gran adaptabilidad a cualquier otro servicio que sea necesario agregar sin perder la precisión que ofrece XML o JSON.

Al diseñar la aplicación con una arquitectura MVC garantizamos la flexibilidad y mantenibilidad antes mencionada y que creemos fundamental para un sistema de este tipo.

****